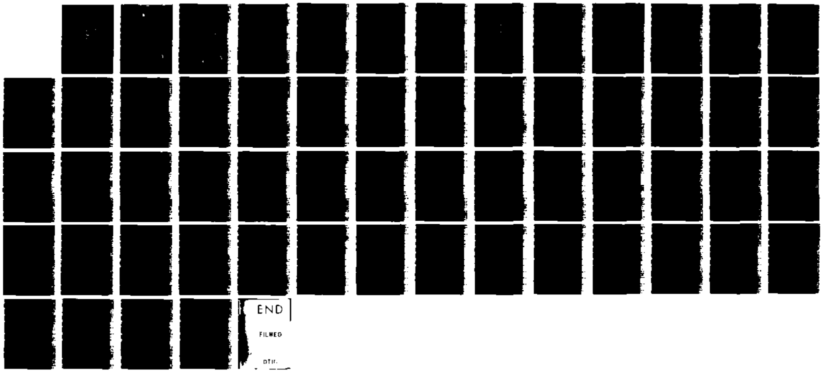
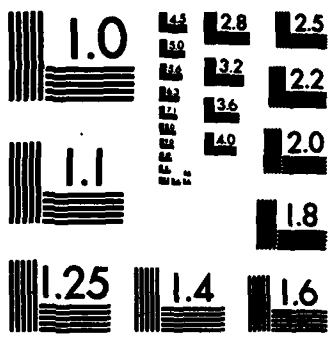
AD-A147 649 FLIGHT SYSTEMS RASTER GRAPHICS SOFTWARE REFERENCE MANUAL(U) AERONAUTICAL RESEARCH LABS MELBOURNE (AUSTRALIA) L N LESTER JUL 84 ARL-SYS-TM-72 1/1

UNCLASSIFIED F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNCLASSIFIED

(12)

ARL-SYS-TM-72

AR-003-939

AD-A147 649

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

AERONAUTICAL RESEARCH LABORATORIES

MELBOURNE, VICTORIA

Systems Technical Memorandum 72

FLIGHT SYSTEMS RASTER GRAPHICS SOFTWARE REFERENCE MANUAL

by

LEWIS N. LESTER

Approved for Public Release

COPY No

JULY 1984

UNCLASSIFIED

84 11 14 007

AR-003-939

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

AERONAUTICAL RESEARCH LABORATORIES

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Systems Technical Memorandum 72

FLIGHT SYSTEMS RASTER GRAPHICS SOFTWARE REFERENCE MANUAL

by

Lewis N. Lester

SUMMARY

Labs. (Melbourne, Australia)

Flight simulation work at Aeronautical Research ~~Laboratories~~ employs raster graphics for both the presentation of visual scenes with limited dynamic content and for the replication of both flight instruments and sensor displays. The raster graphics system consists of an LSI-11/23 microprocessor (functioning as host) and three A.R.L. designed controller units coupled to RGBS monitors, and is connected through the LSI-11/23 to the Flight Simulation Group VAX-11/780 computer. This Memorandum describes the raster graphic system, and four software packages which are available to enable a VAX user to create and manipulate pictures. These packages are: RJRLSI, which runs in the LSI-11/23 for communication; LSILOAD, for loading and executing LSI programs; FLISGRAPH, a set of subroutines for creating and manipulating pictures; and MANPIX, a utility program which provides a versatile range of picture manipulation capabilities through a simple set of commands.

Originator supplied keywords include: Computer graphics, and Computer program documentation.

POSTAL ADDRESS: Director, Aeronautical Research Laboratories, P.O. Box 4331, Melbourne, Victoria, 3001, Australia

CONTENTS

APPENDIX B. PRIMARY BOOTSTRAP FOR USE WITH PROGRAM LSILOAD

DISTRIBUTION

1. INTRODUCTION

The manned flight simulator at Aeronautical Research Laboratories employs raster graphics to emulate various cockpit instruments (such as altimeter and attitude indicator), and for this purpose, a low-cost unit was designed and manufactured at these laboratories. Subsequently, it was found that this raster graphic unit was well-suited to other applications, such as an adjunct in the development of background scenes, and for displaying real-time sequences with small dynamic content (e.g. in air traffic control tower simulation, see [1]).

This unit drives an RGBS monitor and is connected to the Flight Simulation Group VAX-11/780 computer through an LSI-11/23 microcomputer which functions as the graphic unit's host. Several such units can be controlled from the VAX through the one host LSI-11/23. Four software packages are available to enable a VAX user to create and manipulate pictures, namely RJRLSI, LSILOAD, FLISGRAPH and MANPIX. RJRLSI is a program which runs in the host LSI to allow communication between the graphic controllers and programs executing in the VAX. LSILOAD is a VAX program which transmits a user-written LSI program to the LSI and starts its execution. FLISGRAPH is a set of subroutines for picture manipulation, and is designed for use with VAX Pascal, FORTRAN or MACRO programs. MANPIX is an interactive program which enables a user to perform a number of display operations, including the transmission of individual instructions, and the displaying of single pictures or moving sequences.

This Memorandum gives a brief description of the raster graphics unit in Section 2, and describes the communication and loading programs, RJRLSI and LSILOAD, in Section 3. The subroutine package, FLISGRAPH, is presented in Section 4, and Section 5 is a user's guide for the program MANPIX.

2. FLIGHT SYSTEMS RASTER GRAPHICS SYSTEM

The configuration of the system is shown in Figure 1. The host LSI is coupled to the VAX via a direct memory access (DMA) link. There are currently three raster graphic controllers, each of which drives an RGBS monitor. The controllers communicate with the host LSI through separate DMA links.

The controller is the component of the raster graphic unit that was designed and built at ARL. It has 256 colours, each of which contains an 8-bit green field, a 7-bit red field and a 6-bit blue field. This colour "map" can be set up and changed by program control. In addition, up to four of these maps (each is termed a "quadrant") can be stored for different applications (changing quadrants alters the colours in the whole picture - not just components yet to be drawn). The controller also has eight "planes" each of 256K (512 x 512) bits of RAM, capable of storing an 8-bit colour map address for each pixel in a 512 x 512 picture array.

The instruction set for the controller has various features including incremental moves and operation in dump mode. The former allows the fill-in of a specified number of pixels (in any one of the eight cardinal directions) in one instruction. Dump mode operation allows the transfer of part or all of a row or column of pixels between the controller and LSI memories, in either direction. Full details of the instruction set are presented in Appendix A.

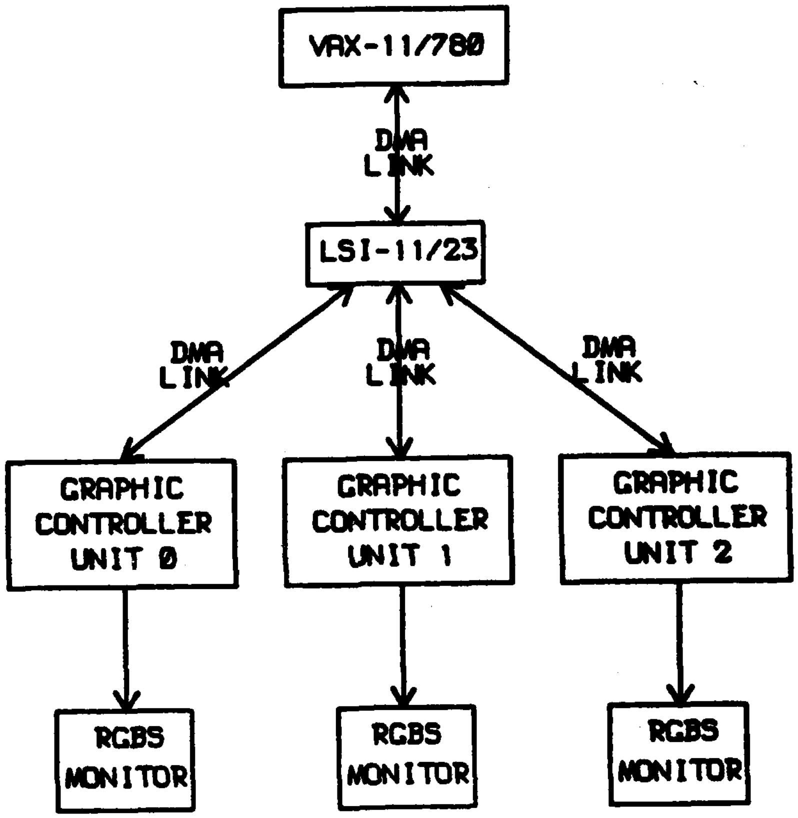


FIGURE 1

Raster Graphics System Configuration

3. HOST LSI COMMUNICATION AND LOADING

3.1 Communication - LSI Program RJRLSI

The transmission of instructions to each graphic controller is achieved by programming the host LSI to send such instructions via the appropriate DMA link. Since the VAX has a very versatile program debugging facility while that of the LSI is extremely limited, it was decided that the host LSI program should be as simple as possible, with the VAX handling the bulk of instruction preparation and formatting.

The program RJRLSI is written in PDP-11 MACRO and performs four basic functions:

(i) selects a controller as current for receiving all future instructions until another controller is selected

(ii) accumulates in a buffer the graphic instructions sent by the VAX for the current controller (a separate buffer is maintained for each controller)

(iii) transmits the accumulated instructions to the current graphic controller, either upon request from the VAX or when its buffer is full

(iv) transmits pixel data to the VAX following a request for the execution of a dump mode read-back operation.

The manner in which RJRLSI operates is to perform some initialisation, and then to wait for signals from the VAX. RJRLSI initialises all graphic controllers by setting their colour map quadrants to zero, loading into each a colour map comprising 256 green intensity levels, clearing the screens, and setting X and Y pixel coordinates to zero. It then executes a loop waiting for a signal from the VAX that a data word is ready for transmission. RJRLSI then copies that word, and checks its value to determine subsequent action as follows:

(i) value = 120000 (octal), 120001 (octal) or 120002 (octal), interpret the value as a command and set the current controller number to 0, 1 or 2 according to the two least significant bits (note that these 16-bit values are all negative when interpreted as signed integers)

(ii) value = n, where $n > 0$, interpret n as an integer and accept n words from the VAX, accumulating them in the current graphic controller's buffer

(iii) value = 0, transmit the accumulated instructions (if any) to the current graphic controller

(iv) value = n, where $n < 0$ and not one of the values mentioned in (i) above, interpret the nine least significant bits as an unsigned integer quantity, m, and send a dump mode instruction to the current controller to read back m pairs of pixels, in the +X direction if the second most significant bit (bit 14) of n is clear, and in the +Y direction if that bit is set. Having read back these values, transmit them to the VAX. (To define the initial pixel for the read-back operation, it is necessary to have previously sent "set coordinate" instructions from the VAX. For more details on the instruction set, see Appendix A.)

3.2 Loading - Program LSILOAD

Since the host LSI has no peripheral storage device, it is necessary after power-up to load and subsequently execute a program which has been stored elsewhere. The function of program LSILOAD is to transfer a program from the VAX to the LSI, and then to start it executing. In order to use LSILOAD, it is necessary to prepare the LSI program in absolute loader format. This can be done by coding in PDP-11 MACRO, using the ".enabl abs" directive, and explicitly setting the value of the program counter (program RJRLSI uses this approach). If such a program is then assembled using the VAX-11 MACRO/RSX assembler, the resulting object file will be in the absolute loader format.

To load an LSI program from the VAX, the user must firstly load the primary bootstrap into the LSI. The primary bootstrap instructions for transfers using LSILOAD are listed in Appendix B. Once the bootstrap is loaded (but NOT executing), the VAX program LSILOAD is run. The program will immediately respond with

```
DON'T START THE BOOT YET
Enter LSI filename:
```

at which point the user should enter the name of the LSI program to be loaded. LSILOAD will then reply with

```
Now start the primary boot, then enter <CR>
```

The user should then start the execution of the primary bootstrap, and enter a carriage-return. LSILOAD will then send a secondary bootstrap to the LSI, start its execution and transfer the LSI program. When the transfer is complete, the secondary bootstrap will start the execution of the LSI program.

4. PROGRAMMING WITH THE FLISGRAPH PACKAGE

The FLISGRAPH package consists of five VAX-11 MACRO subroutines and a number of VAX-11 FORTRAN subroutines, all of which may be called from VAX-11 Pascal, FORTRAN, or MACRO programs in order to create or alter pictures on the Flight Systems Raster Graphic display.

Operating with FLISGRAPH routines is, in most cases, initialised by a call to GRAPHIC_INIT and terminated by a call to GRAPHIC_FINISH, with all other calls lying between these two. Certain utility routines within the package do not require this discipline, and this fact will be pointed out in their descriptions.

Note that at any one time a user can communicate with only one of the three controller units of the raster graphics system. This unit is referred to as the "current controller". If it is necessary to manipulate the picture on more than one screen, each must be selected in turn as the current controller before changes can be made.

The descriptions of the routines within the FLISGRAPH package will now be presented. Several of these routines have optional arguments, in the sense that, starting with the right-most, one or more can be omitted and a default value will apply. Such arguments will be shown in brackets ([]), with nesting of the brackets where appropriate.

4.1 Initialisation, Selection and Termination Routines

4.1.1 Logical Function GRAPHIC-INIT([code[,error-lim]])

4.1.1.1 Purpose

This routine allocates the DMA link between the VAX and the LSI to the user, sets graphic controller 0 as the current unit, sets the text font number to zero, and disables the inhibit-display switch (see section 4.2.4). This routine must be called before any of the other picture manipulating routines.

4.1.1.2 Arguments

There are two arguments, both of which are optional. However, if the second argument is specified, then the first must also be included.

(1) **code**
Type: integer
Use: This argument specifies the action to be taken if the DMA link is allocated to another user, resulting in the graphic system being unavailable. This gives a user four options so that computer time spent in picture generation is not subsequently wasted when the picture cannot be displayed immediately. The values that **code** can have, and the subsequent actions are as follows:

0 - dialogue - the user is asked whether he wants to exit, try again, or be told who is currently using the graphic system

1 - terminate - stop the program if the graphic system is unavailable

2 - no action - attempt to allocate and set the returned function value according to success or failure

3 - wait - keep trying to allocate at 1 minute intervals until successful

If **code** is omitted, or specified as any other value, it is treated as having been specified as 0.

(2) **error-lim**
Type: integer
Use: With the spot positioning, text and vector drawing routines (described below in sections 4.4 and 4.5), an attempt may be made to set coordinates outside the 512 x 512 pixels of the picture. Argument **error-lim** is used to specify the maximum number of times that such an error will be reported at the terminal. If **error-lim** is specified as negative, no limit will be imposed, and if omitted, a default value of 5 is used.

4.1.1.3 Function Return

GRAPHIC-INIT will return true if the DMA link was successfully allocated, and false if it was not. The function return is only useful when argument **code** is specified as 2 - for other values of **code**, pre-determined actions are executed in the event of the graphic system being allocated to another user. Note that GRAPHIC-INIT can be called as a subroutine rather than as a function when **code** $\neq 2$.

4.1.2 Subroutine SCREEN_SELECT(screen_number)

4.1.2.1 Purpose

This routine changes the number of the current graphic controller. All instructions sent by graphics routines are placed into the buffer of the controller selected by this routine, and this controller remains current until altered by another call to SCREEN_SELECT (or SET_CONTROLLER, see section 4.1.4). As mentioned above, controller 0 is initially selected by GRAPHIC_INIT.

4.1.2.2 Arguments

SCREEN_SELECT has one argument:

(1) **screen_number**
Type: integer
Use: This argument is the number of the controller which is to become the current controller, and should be 0, 1 or 2.

4.1.2.3 Errors

A fatal error results if SCREEN_SELECT is called without a preceding call to GRAPHIC_INIT, or if it is called with an argument whose value is not 0, 1 or 2.

4.1.3 Integer Function WHICH_SCREEN()

4.1.3.1 Purpose

This function returns to the user the number of the current controller.

4.1.3.2 Arguments

WHICH_SCREEN has no arguments.

4.1.3.3 Function Return

WHICH_SCREEN will return 0, 1 or 2 according to whichever is the current controller.

4.1.3.4 Errors

A fatal error results if WHICH_SCREEN is called without a preceding call to GRAPHIC_INIT.

4.1.4 Subroutine SET_CONTROLLER(screen_number[,clear[,no_reset]])

4.1.4.1 Purpose

This routine changes the number of the current graphic controller, and then initialises the controller's x and y coordinates to zero, sets the memory mask to enable all planes, resets the scroll offset, enables picture displaying and sets the colour map read and write quadrants to zero. The user can optionally clear the screen, and suppress this initialisation.

4.1.4.2 Arguments

SET_CONTROLLER has three arguments, the last two being optional:

(1) **screen_number**
Type: integer
Use: This argument is the number of the controller which is to become the current controller, and should be 0, 1 or 2.

(2) **clear**

Type: logical

Use: This argument is a flag to indicate whether the new current controller screen should be cleared. If **clear** is specified as true, all pixels are set to colour 0 (usually this is set up in the map as black). If **clear** is specified as false, or omitted, no such action is taken.

(3) **no_reset**

Type: logical

Use: This argument is a flag to indicate whether or not the above-mentioned initialisation should be suppressed. If **no_reset** is specified as true, the initialisation is suppressed. If it is specified as false, or omitted, the initialisation is performed.

4.1.4.3 Errors

A fatal error results if SET_CONTROLLER is called without a preceding call to GRAPHIC_INIT, or if it called with its first argument outside the range of 0-2.

4.1.5 Subroutine GRAPHIC_FINISH([clear])

4.1.5.1 Purpose

This routine releases the DMA link between the VAX and the LSI making it available to other users. If any spot positioning errors were detected, the total count of these will be reported at the user's terminal. After a GRAPHIC_FINISH call, GRAPHIC_INIT must be called to re-initialise the graphic system if further picture processing is to be done.

4.1.5.2 Arguments

GRAPHIC_FINISH has one optional argument:

(1) **clear**

Type: logical

Use: This argument is a flag to indicate whether or not the picture on the current screen should be zeroed before the graphic system is relinquished. If specified as true, all pixels are set to colour 0. If **clear** is specified as false, or is omitted, the picture is left on the screen.

4.1.5.3 Errors

A fatal error results if GRAPHIC_FINISH is called without a preceding call to GRAPHIC_INIT.

4.2 Data Transmission Routines

4.2.1 Subroutine SEND_DATA(values,num_values)

4.2.1.1 Purpose

This routine is called whenever it is necessary to transmit graphics instructions to the current controller. Each call to SEND_DATA places instructions into a buffer reserved in the VAX for that controller. When this buffer is full, its contents are transmitted via the DMA link to a corresponding buffer in the host LSI. When that buffer is full, the LSI transfers its contents to the current controller. Alternatively, transmission of a partially filled buffer can be forced by calling subroutine DISPLAY (see section 4.2.2). Thus, a user can send several groups of instructions,

and have transmission of these to the current controller delayed until all groups have been accumulated.

4.2.1.2 Arguments

SEND_DATA has two arguments:

(1) **values**
Type: integer*2 array or logical*2 array (note the "*2" specification)

Use: This array specifies the data to be sent to the graphic system host LSI.

(2) **num-values**
Type: integer
Use: This argument specifies the number of words in the **values** array.

4.2.2.3 Errors

A fatal error results if SEND_DATA is called without a preceding call to GRAPHIC_INIT.

4.2.2 Subroutine DISPLAY

4.2.2.1 Purpose

This routine transmits to the current controller any instructions which have accumulated in the VAX or LSI buffers through calls to SEND_DATA (see section 4.2.1). If there are no instructions in either buffer, no action is taken.

4.2.2.2 Arguments

DISPLAY has no arguments.

4.2.2.3 Errors

A fatal error results if DISPLAY is called without a preceding call to GRAPHIC_INIT.

4.2.3 Subroutine FETCH_DATA(values,num-values)

4.2.3.1 Purpose

This routine is used when data transmission is required from the graphic system's host LSI to the VAX. This will normally arise when part or all of the picture of the current controller is to be read back. Note that the LSI will not transmit data until it has received the command to do so, and hence a call to FETCH_DATA must be preceded by a call to SEND_DATA which sends a single word containing a negative integer as described in section 3.1. This will cause the host LSI to start transmission of pixel data, and FETCH_DATA will then receive this data and store the pixels in pairs as 16 bit (integer*2) words.

4.2.3.2 Arguments

FETCH_DATA has two arguments:

(1) **values**
Type: integer*2 array or logical*2 array (note the "*2" specification)

Use: This array receives the data to be sent to the VAX by the graphic unit host LSI.

(2) num_values
Type: integer
Use: This argument specifies the number of words to be stored in the values array. This will correspond to the number of pixel pairs to be read from the graphic controller's memory.

4.2.3.3 Errors

A fatal error results if FETCH_DATA is called without a preceding call to GRAPHIC_INIT.

4.2.4 Subroutine INHIBIT

4.2.4.1 Purpose

Each transmission of instructions from the VAX to the current controller incurs an overhead in VAX processing. This overhead does not depend upon the number of instructions being sent, and hence it is more efficient to send a single large block of instructions rather than several smaller blocks. Several routines in the FLISGRAPH package (e.g. SET_COLOUR (section 4.3.3) or MOVE (section 4.4.4)) involve the transmission of a very small number of instructions, and these are normally sent as they are generated by a call to DISPLAY within each routine. This has an immediate effect upon the picture on the current screen, at some cost in efficiency. If it is not necessary to observe the result of each routine as it is called, efficiency can be improved by a call to INHIBIT. This sets a flag within the FLISGRAPH package so that these small groups of instructions are not transmitted until the user explicitly requests it with a call to DISPLAY.

4.2.4.2 Arguments

INHIBIT has no arguments.

4.2.5 Subroutine UN_INHIBIT

4.2.5.1 Purpose

This routine disables the inhibit-display switch within the FLISGRAPH package, so that all routines send their graphics instructions as they are generated. This routine has the opposite effect of INHIBIT (section 4.2.4).

4.2.5.2 Arguments

UN_INHIBIT has no arguments.

4.3 Raster Graphic Controller Operation Routines

4.3.1 General

These routines implement directly several of the graphic controller's instructions (the full instruction set appears in Appendix A). Note that all these routines call SEND_DATA and hence a fatal error will result if there has been no previous call to GRAPHIC_INIT.

4.3.2 Subroutine SET_COLOUR(colour)

4.3.2.1 Purpose

This routine sends the appropriate instruction to the current graphic controller to change the value in the controller's colour code register. This colour is then used in all future instructions which write the controller memory until changed again.

4.3.2.2 Arguments

SET_COLOUR has one argument:

(1) **colour**
Type: integer
Use: This argument specifies the value of the new colour, and should lie between 0 and 255 inclusive. If **colour** is outside this range, the least significant eight bits are used.

4.3.3 Subroutine SET_MASK(mask)

4.3.3.1 Purpose

This routine sends the appropriate instruction to the current graphic controller to change the value in the controller's mask register. This register is used to determine which memory planes can be written. Setting one of the mask bits to zero prevents modification of the corresponding bit plane, and results in a sub-set of the colour map being used.

4.3.3.2 Arguments

SET_MASK has one argument:

(1) **mask**
Type: integer
Use: This argument specifies the value of the new mask, and should lie between 0 and 255 inclusive. If **mask** is outside this range, the least significant eight bits are used to determine which memory planes are write-enabled.

4.3.4 Subroutine SET_QUADRANT(display_quad,write_quad)

4.3.4.1 Purpose

This routine sends the appropriate instructions to the current graphic controller to change the display and write colour map quadrants. Selection of a display quadrant determines which of the four stored colour maps (see section 2.0) will be used, while selection of a write quadrant determines which of the four maps will be altered by future "load-colour-map" instructions.

4.3.4.2 Arguments

SET_QUADRANT has two arguments:

(1) **display_quad**
Type: integer
Use: This argument specifies the new display quadrant number, and should lie between 0 and 3 inclusive. If **display_quad** is outside this range, no action is taken, so that this routine can be used to select write quadrant only.

(2) **write_quad**
Type: integer
Use: This argument specifies the new write quadrant number, and should lie between 0 and 3 inclusive. If **write_quad** is outside this range, no action is taken, so that this routine can be used to select display quadrant only.

4.3.4 Subroutine SET_SCREEN(colour)

4.3.4.1 Purpose

This routine sends the instruction to the current graphic controller to colour all pixels of the screen in a particular value. Since most colour maps are set up with colour 0 being black, this routine can be used to clear the screen by calling it with an argument of 0.

4.3.5.2 Arguments

SET_SCREEN has one argument:

(1) **colour**
Type: integer

Use: This argument specifies the value of the colour to which all pixels will be set, and should lie between 0 and 255 inclusive. If colour is outside this range, the least significant eight bits are used.

4.3.6 Subroutine SCROLL(x,y)

4.3.6.1 Purpose

This routine sends the appropriate instruction to the current graphic controller to scroll the picture by one pixel or to reset the scroll offset.

4.3.6.2 Arguments

SCROLL has two arguments:

(1) **x**
Type: integer
Use: This argument defines the scroll to be performed in the horizontal direction. If positive, a +X scroll is effected, if negative, a -X scroll is effected. If zero, no X scroll takes place.

(2) **y**
Type: integer
Use: This argument defines the scroll to be performed in the vertical direction. If positive, a +Y scroll is effected, if negative, a -Y scroll is effected. If zero, no Y scroll takes place. Note that if x and y are both zero, a scroll reset is effected.

4.3.7 Subroutine LOAD_COLOURS(count,addresses,red,green,blue)

4.3.7.1 Purpose

This routine sends instructions to the current graphic controller to load any or all of the elements of the colour map with a set of red, green and blue components. The quadrant affected is the one selected by the most recent write quadrant instruction.

4.3.7.2 Arguments

LOAD_COLOURS has five arguments:

(1) **count**
Type: integer
Use: This argument specifies the number of elements of the colour map that are to be changed.

(2) **addresses**
Type: integer array
Use: This array of count elements specifies the addresses in the colour map which are to be changed.

(3) **red**
Type: integer array
Use: This array of count elements specifies the new values of the red components of the colours in the map. The red values should lie between 0 and 127 inclusive. If any value is outside this range, the least significant seven bits are used.

(4) **green**
Type: integer array
Use: This array of count elements specifies the new values of the green components of the colours in the map. The green values should lie between 0 and 255 inclusive. If any value is outside this range, the least significant eight bits are used.

(5) **blue**
Type: integer array
Use: This array of count elements specifies the new values of the blue components of the colours in the map. The blue values should lie between 0 and 63 inclusive. If any value is outside this range, the least significant six bits are used.

4.3.8 Subroutine LOAD_MAP_FILE(quadrant,filename[,details])

4.3.8.1 Purpose

This routine loads a colour map with a precomputed set of values stored in a formatted binary file. It is assumed that colour maps are generated from a set of hue, saturation and intensity values since most observers are better able to identify colours by these rather than by their primary colour components. However, the hue, saturation and intensity values must be converted to red, green and blue components for loading into the graphic controller, and hence the format for the colour map file includes information describing the generation in addition to the actual colours. Since the saturation of a whole scene is more generally dependent on external conditions (e.g. fog) than on entities within the scene, it is not unreasonable to define as constant the saturation for a colour map. The remainder of the map can be specified by choosing the number of intensity levels for each colour, say N, which then allows 256/N hues. This definition can be pictured by treating colour space as a pair of cones joined base-to-base, with intensity varying along the axis, saturation increasing radially, and with hue being defined by an angular displacement around the cone. Generally, red is defined to be at 0 degrees, green at 120 degrees and blue at 240 degrees, with varying hues in between. As intensity becomes extreme, the colour is perceived as nearly white or nearly black, with little hue and saturation components.

A further complication is that the eye has a non-linear perception of intensity, so it is necessary to introduce a correction factor (the "gamma" correction) to allow for this. The gamma factors (one for each primary colour) used in producing the map are also stored in the map file.

The map file must be a FORTRAN binary file comprising one record set out as follows:

n = number of colours to be set up	integer*4
map addresses (between 0 and 255)	logical*1 array(1:n)
red components (between 0 and 127)	logical*1 array(1:n)
green components (between 0 and 255)	logical*1 array(1:n)
blue components (between 0 and 63)	logical*1 array(1:n)
angular separation between hues (degrees)	real*4
saturation (between 0 and 1)	real*4
number of intensity levels	real*4
red gamma correction value	real*4
green gamma correction value	real*4
blue gamma correction value	real*4

4.3.8.2 Arguments

LOAD_MAP_FILE has three arguments, the third being optional:

(1) **quadrant**
Type: integer
Use: This argument specifies the quadrant into which the colour map is to be loaded, and should be between 0 and 3 inclusive. This quadrant becomes the current one for all future instructions. If **quadrant** is out of range, no quadrant change takes place and the current quadrant is used.

(2) **filename**
Type: character string
Use: This argument specifies the name of the file containing the colour map to be loaded. It is assumed that this file exists in the directory defined by the logical name MAP:. If this file does not exist in the MAP: directory, the user's directory will be searched for the file.

(3) **details**
Type: real array of 6 elements
Use: If this argument is specified, the last 6 entries in the file of the colour map will be returned as the 6 array elements.

4.3.8.3 Errors

A fatal error will result if the file does not exist in the MAP: directory or in the user's directory, or if the file format does not match that described above.

4.4 Spot Positioning and Vector Drawing Routines

4.4.1 General

The routines in this section keep track of the controller's x and y coordinate values ("spot position") and each call checks whether or not the requested move will result in the spot position being off screen. If this is the case, no action is taken, but the number of spot positioning errors is incremented. If the accumulated number of off-screen errors is less than the limit set up in the call to GRAPHIC_INIT (see section 4.1.1), a warning message will be given. The total number of spot positioning errors will always be reported to the user when GRAPHIC_FINISH is called (see section 4.1.5). Note that the origin is the bottom left-hand corner of the screen.

Note also that these routines call SEND_DATA and hence a fatal error will result if there has been no previous call to GRAPHIC_INIT.

4.4.2 Subroutine DRAW_SPOT(x,y[,colour])

4.4.2.1 Purpose

This routine draws a spot at the point whose coordinates are specified in the call, provided that that point is on the screen. The spot is drawn by first setting the current controller colour, then setting its x and y position values and writing that pixel.

4.4.2.2 Arguments

DRAW_SPOT has three arguments, the third being optional:

(1) **x**
Type: integer
Use: This argument specifies the X-coordinate of the spot to be drawn, and should be between 0 and 511 inclusive.

(2) **y**
Type: integer
Use: This argument specifies the Y-coordinate of the spot to be drawn, and should be between 0 and 511 inclusive.

(3) **colour**
Type: integer
Use: This argument specifies the colour in which the spot is to be drawn, and should be between 0 and 255 inclusive. If **colour** is omitted, the current colour is used; if specified, it replaces the current colour. If **colour** is out of range, the least significant eight bits are used.

4.4.3 Subroutine DRAW_SPOT_REL(x,y[,colour])

This routine does the same as the DRAW_SPOT routine but the x and y arguments are interpreted as coordinates relative to the current spot position rather than as absolute coordinates.

4.4.4 Subroutine MOVE(x,y)

4.4.4.1 Purpose

This routine sets the current controller's x and y position values to those specified in the call, provided that the desired position is on the screen. Note that it does not draw anything on the screen. It is used for positioning vectors or text.

4.4.4.2 Arguments

MOVE has two arguments:

(1) **x**
Type: integer
Use: This argument specifies the X-coordinate of the new spot position, and should be between 0 and 511 inclusive.

(2) **y**
Type: integer
Use: This argument specifies the Y-coordinate of the new

spot position, and should be between 0 and 511 inclusive.

4.4.5 Subroutine REL_MOVE(x,y)

This routine does the same as the MOVE routine but the x and y arguments are interpreted as coordinates relative to the current spot position rather than as absolute coordinates.

4.4.6 Subroutine VECTOR(x,y[,colour])

4.4.6.1 Purpose

This routine checks that the point (x,y) is on the screen, and if so, moves the spot to that point and draws the vector between the last position of the spot and (x,y) in the specified colour.

4.4.6.2 Arguments

VECTOR has three arguments, the third being optional:

(1) **x**
Type: integer
Use: This argument specifies the X-coordinate of the end point of the vector, and should be between 0 and 511 inclusive.

(2) **y**
Type: integer
Use: This argument specifies the Y-coordinate of the end point of the vector, and should be between 0 and 511 inclusive.

(3) **colour**
Type: integer
Use: This argument specifies the colour in which the vector is to be drawn, and is specified in exactly the same manner as the colour argument in DRAW_SPOT.

4.4.7 Subroutine REL_VECTOR(x,y[,colour])

This routine does the same as the VECTOR routine but the x and y arguments are interpreted as coordinates relative to the current spot position rather than as absolute coordinates.

4.4.8 Subroutine H_VECTOR(x_end[,colour])

4.4.8.1 Purpose

This routine is a direct implementation of the graphic controller's incremental move instruction, and draws a horizontal vector from ONE PIXEL AFTER the current spot position (in the appropriate direction) to the end pixel specified in the call, provided that the end point is on the screen. A call to this routine would usually be preceded by a call to DRAW_SPOT; these routines used in this way provide the fastest way of drawing horizontal vectors.

4.4.8.2 Arguments

H_VECTOR has two arguments, the second being optional:

(1) **x_end**
Type: integer
Use: This argument specifies the X-coordinate of the end point of the vector, and should be between 0 and 511 inclusive.

(2) **colour**
Type: integer
Use: This argument specifies the colour in which the vector is to be drawn, and is specified in exactly the same manner as the colour argument in DRAW_SPOT.

4.4.9 Subroutine V_VECTOR(y_end[,colour])

This routine draws a vertical vector in exactly the same manner as H_VECTOR draws a horizontal vector. Combined with DRAW_SPOT, it is the fastest way to draw vertical vectors.

4.4.9.1 Arguments

V_VECTOR has two arguments, the second being optional:

(1) **y_end**
Type: integer
Use: This argument specifies the Y-coordinate of the end point of the vector, and should be between 0 and 511 inclusive.

(2) **colour**
Type: integer
Use: This argument specifies the colour in which the vector is to be drawn, and is specified in exactly the same manner as the colour argument in DRAW_SPOT.

4.4.10 Subroutine LOCATE_POINT (cross_colour,x,y[,xstart[,ystart]])

4.4.10.1 Purpose

This routine enables a user to locate interactively a point on the current screen. The user is asked to specify moves to position a set of "cross-hairs" about the screen until it is at the desired point. Moves are specified interactively as Un, Dn, Ln or Rn for n pixel-steps upwards, downwards, left or right. Once the desired point is reached, the user specifies a zero rather than a move and the spot coordinates are returned in the subroutine argument list.

4.4.10.1 Arguments

LOCATE_POINT has five arguments, the last two being optional:

(1) **cross_colour**
Type: integer
Use: This argument specifies the colour of the "cross-hairs", and should lie between 0 and 255 inclusive. If **cross_colour** is outside this range, the least significant eight bits are used.

(2) **x**
Type: integer
Use: This argument is an output parameter and contains the x-coordinate of the located point.

(3) **y**
Type: integer
Use: This argument is an output parameter and contains the y-coordinate of the located point.

(4) **xstart**
Type: integer
Use: This argument specifies the intial x-coordinate of the centre of the "cross-hairs". If omitted, a value of 256 (screen centre) is used.

(5) **ystart**
Type: integer
Use: This argument specifies the intial y-coordinate of the centre of the "cross-hairs". If omitted, a value of 256 (screen centre) is used.

4.5 Text Routines

4.5.1 General

The routines in this section provide a text drawing capability. The FLISGRAPH package contains two text fonts, with font 0 being a set of characters 9 pixels high and 5 pixels wide, and font 1 being an 11 pixel high by 6 pixel wide set. For text strings, characters are drawn with a spacing of two pixels. All 128 ASCII codes are interpreted. Carriage return, linefeed, backspace and tab are interpreted as would be expected, whereas all other control codes (in the range 0 to 31) are represented by drawing, enclosed in a rectangle, the character whose ASCII code is the control code plus 64.

4.5.2 Subroutine FONT(font_number)

4.5.2.1 Purpose

This routine sets an internal value in the FLISGRAPH package to determine which font is to be used in future character drawing routines. Note that a call to routine GRAPHIC_INIT always resets the font number to 0.

4.5.2.2 Arguments

FONT has one argument:

(1) **font_number**
Type: integer
Use: This argument specifies the font which will be used for all future character drawing, and should be either 0 (9 x 5 set) or 1 (11 x 6) set.

4.5.2.3 Errors

A warning is given if a font number other than 0 or 1 is specified as the argument. In this case, the font number is unaltered.

4.5.3 Subroutine DRAW_CHAR(ch,orientation,success)

4.5.3.1 Purpose

DRAW_CHAR is the basic routine for drawing characters, and is called by all other text routines. It checks that the character will fit on the screen, and if it will, the character is drawn in the current colour, with its lower left corner at the current spot position. The spot position is then updated to the lower left corner of the next character space. The text font used is that specified in the most recent call to routine FONT (or 0 if FONT has not been called explicitly).

4.5.3.2 Arguments

DRAW_CHAR has three arguments:

(1) **ch**
Type: character*1
Use: This argument specifies the character to be drawn.

(2) **orientation**
Type: integer
Use: This argument specifies the orientation at which the character is to be drawn, and should be between 0 and 3 inclusive. If **orientation** is outside this range, the least significant 2 bits are used. The value of **orientation** is interpreted as follows:

0: The character is drawn without rotation.

1: The character is rotated 90 degrees counter-clockwise before it is drawn.

2: The character is rotated 180 degrees counter-clockwise before it is drawn (i.e. it appears inverted).

3: The character is rotated 270 degrees counter-clockwise before it is drawn.

(3) **success**
Type: logical
Use: This argument is set by DRAW_CHAR and indicates whether or not the character was drawn successfully. Failure occurs when the character will not fit completely on the screen.

4.5.3.3 Errors

If DRAW_CHAR detects that the character will not fit on the screen, it takes no action other than to increment the number of spot positioning errors. If the accumulated number of off-screen errors is less than the limit set up in the call to GRAPHIC_INIT (see section 4.1.1), a warning message will be given.

This routine calls SEND_DATA and hence a fatal error will result if there has been no previous call to GRAPHIC_INIT.

4.5.4 Subroutine DRAW_TEXT (string[,nchars[,orientation[,colour]]])

4.5.4.1 Purpose

DRAW_TEXT calls subroutine DRAW_CHAR to draw each of the characters in a string (in the current font) onto the graphic screen, with the first character's lower left corner at the current spot position. If the entire string will not fit on the screen, only those characters that fit completely will be drawn. DRAW_TEXT updates the spot position to the lower left corner of the next character space.

4.5.4.2 Arguments

DRAW_TEXT has four arguments, with the last three being optional. However, if a particular optional argument is specified, then all arguments to its left in the calling sequence must also be specified.

(1) **string**
Type: character string
Use: This argument specifies the string of characters to be drawn.

(2) **nchars**
Type: integer
Use: This argument specifies the number of characters in the string to be drawn. If **nchars** is omitted, or is specified as negative, zero, or greater than the length of the string, then the entire string is drawn.

(3) **orientation**
Type: integer
Use: This argument specifies the orientation at which the characters in the string are to be drawn, and is specified in exactly the same manner as the **orientation** argument in DRAW_CHAR. If **orientation** is omitted, then a value of 0 (i.e. no rotation) is used.

(4) **colour**
Type: integer
Use: This argument specifies the colour in which the string is to be drawn, and should be between 0 and 255 inclusive. If **colour** is omitted, the current colour is used; if specified, it replaces the current colour. If **colour** is out of range, the least significant eight bits are used.

4.5.4.3 Errors

Since DRAW_TEXT uses subroutine DRAW_CHAR, it will be subject to any errors detected by DRAW_CHAR.

4.5.5 Subroutine BIG_CHAR(ch,success)

4.5.5.1 Purpose

BIG_CHAR is an entry point in subroutine DRAW_CHAR and draws double-sized characters (in the current font) by drawing each pixel in the basic character as a 2 x 2 square of pixels, provided that the character will fit completely on the screen. The character is drawn in the current colour, with its lower left corner at the current spot position. The spot position is then updated to the lower left corner of the next character space. This routine does not allow characters to be rotated, and draws all control characters as spaces.

4.5.5.2 Arguments

BIG_CHAR has two arguments:

(1) **ch**
Type: character*1
Use: This argument specifies the character to be drawn.

(2) **success**
Type: logical
Use: This argument is set by BIG_CHAR and indicates whether or not the character was drawn successfully. Failure occurs when the character will not fit completely on the screen.

4.5.5.3 Errors

BIG_CHAR detects the same errors that DRAW_CHAR detects, and responds to them in the same manner.

4.5.6 Subroutine BIG_TEXT(string[,nchars[,colour]])

4.5.6.1 Purpose

BIG_TEXT calls subroutine BIG_CHAR to draw each of the characters in a string at double size (in the current font), with the first character's lower left corner at the current spot position. If the entire string will not fit on the screen, only those characters that fit completely will be drawn. BIG_TEXT updates the spot position to the lower left corner of the next character space.

4.5.6.2 Arguments

BIG_TEXT has three arguments, with the last two being optional. However, if the third argument is to be specified, then the second must also be specified.

(1) **string**
Type: character string
Use: This argument specifies the string of characters to be drawn.

(2) **nchars**
Type: integer
Use: This argument specifies the number of characters in the string to be drawn, and is interpreted in exactly the same manner as the **nchars** argument in routine DRAW_TEXT.

(3) **colour**
Type: integer
Use: This argument specifies the colour in which the string is to be drawn, and is interpreted in exactly the same manner as the **colour** argument in routine DRAW_TEXT.

4.5.6.3 Errors

Since BIG_TEXT uses subroutine BIG_CHAR, it will be subject to any errors detected by BIG_CHAR.

4.6 Picture Transmission Routines

4.6.1 Subroutine DRAWPIC(picture,left,right,bottom,top)

4.6.1.1 Purpose

This routine transmits a picture by copying an array from VAX memory into the current graphic controller's memory. The array represents a rectangular section of the screen, and contains one byte for each pixel to be coloured. DRAWPIC sets up the pixel colour information as a sequence of dump mode restore instructions which are transmitted to the graphic controller with SEND_DATA.

4.6.1.2 Arguments

DRAWPIC has five arguments:

(1) **picture**
Type: logical*1 array
Use: This array specifies the colour of each pixel to be

changed. Note that this array must be dimensioned properly - its declaration must be of the form

logical*1 picture(N1:N2,M1:M2)

In general, N1=**left**, N2=**right**, M1=**bottom**, M2=**top** which are the other arguments to DRAWPIC. However, it is only necessary that the size and shape of the rectangle as specified in the dimensions match exactly the size and shape as specified by the other arguments, i.e., N2-N1=**right**-**left**, and M2-M1=**top**-**bottom**. The effect of varying N1, N2, M1, M2 from **left**, **right**, **bottom**, top respectively is to translate the picture before it is drawn on the screen.

(2) **left**
(3) **right**
(4) **bottom**
(5) **top**
Type: integer
Use: These arguments specify the limits of the rectangle to be drawn, and have a close correlation with the dimensions of the **picture** array as described above.

4.6.1.3 Errors

This routine calls SEND_DATA and hence a fatal error will result if there has been no previous call to GRAPHIC_INIT.

4.6.2 Subroutine READBACK (picture,x_dimension,left,right,bottom,top)

4.6.2.1 Purpose

This routine copies a rectangular section from the current graphic controller's memory into an array in VAX memory, which contains one byte for each pixel to be read back. READBACK obtains the information by commanding the LSI program RJRLSI to send a sequence of dump mode save instructions to the graphic controller. The VAX then fetches the pixel information with FETCH_DATA.

4.6.2.2 Arguments

READBACK has six arguments:

(1) **picture**
Type: logical*1 array
Use: This array receives the pixel data, and hence must be dimensioned large enough to have (at least) one element for each pixel to be read. That is, its total length must not be less than (**right**-**left**+1)*(**bottom**-**top**+1).

(2) **x_dimension**
Type: integer
Use: This argument specifies the actual length of a row of the picture as dimensioned in the calling program. For example, if it is required to read back that rectangle of the picture bounded by (0,0), (0,20), (10,20) and (10,0) into an array defined similarly, the following code could be used:

```
logical*1 picture(0:10,0:20)
call readback(picture,11,0,10,0,20)
```

On the other hand, if it is required to read that same rectangle into a full picture array, the following code could be used:

```
logical*1 picture(0:511,0:511)
call readback(picture,512,0,10,0,20)
```

(3) **left**
(4) **right**
(5) **bottom**
(6) **top**
Type: integer
Use: These arguments specify the limits of the rectangle to be read back. In the above example, **left**=0, **right**=10, **bottom**=0 and **top**=20.

4.6.2.3 Errors

This routine calls FETCH_DATA and hence a fatal error will result if there has been no previous call to GRAPHIC_INIT.

4.6.3 Subroutine DRAW_BOX(left,right,bottom,top,[colour])

4.6.3.1 Purpose

This routine draws a coloured rectangle on the current screen in a specified colour.

4.6.3.2 Arguments

DRAW_BOX has five arguments, the last one being optional:

(1) **left**
(2) **right**
(3) **bottom**
(4) **top**
Type: integer
Use: These arguments specify the limits of the rectangle to be drawn.

(5) **colour**
Type: integer
Use: This argument specifies the colour in which the rectangle is to be drawn, and should be between 0 and 255 inclusive. If **colour** is omitted, the current colour is used; if specified, it replaces the current colour. If **colour** is out of range, the least significant eight bits are used.

4.6.3.3 Errors

This routine calls SEND_DATA and hence a fatal error will result if there has been no previous call to GRAPHIC_INIT.

4.6.4 Subroutine DRAW_MAP_BOXES(text_colour)

4.6.4.1 Purpose

This routine allows a user to view the colour map of the current display quadrant of the current controller. It fills the screen with 256 rectangles - one in each colour of the map, and then writes the colour number

of each (in hexadecimal radix) onto the rectangle. The user specifies the colour of this text.

4.6.4.2 Arguments

DRAW_MAP_BOXES has one argument:

(1) **text_colour**
Type: integer
Use: This argument specifies the colour of the text which is written into each rectangle, identifying its colour number, and should be between 0 and 255 inclusive.

4.6.4.3 Errors

This routine calls SEND_DATA and hence a fatal error will result if there has been no previous call to GRAPHIC_INIT.

4.7 Utility Routines

4.7.1 General

These routines provide some capabilites for the translation of pictures from pixel colour data to graphic controller instructions and vice versa. These routines do not interact with the raster graphic controller and hence it is not necessary to call GRAPHIC_INIT prior to their use.

4.7.2 Subroutine DECIPHER(instructions,num-inst,log-unit,seq-num)

4.7.2.1 Purpose

This routine decodes 16-bit graphic controller instruction words. The output can be directed to the user's terminal or to a file for printing.

4.7.2.2 Arguments

DECIPHER has four arguments:

(1) **instructions**
Type: integer*2 array or logical*2 array
Use: This argument specifies the array containing the sequence of instructions to be decoded.

(2) **num_inst**
Type: integer
Use: This argument specifies the length of the array **instructions**, i.e. the number of instructions to be decoded.

(3) **log_unit**
Type: integer
Use: This argument specifies a FORTRAN logical unit number on which the output file has been opened. If **log_unit** is specified as zero or negative, output is directed to the user's terminal.

(4) **seq_num**
Type: integer
Use: This argument specifies the sequence number that will prefix the first line of output from this routine. Each instruction word is deciphered into one line of output, comprising the sequence number, the instruction and its translation. **seq_num** is incremented after each instruction is decoded, so that several calls to DECIPHER (without the user altering

seq_num) will result in the sequence numbers being continuous.

4.7.2.3 Errors

DECIPHER detects no errors. However, a warning is issued if an instruction is not recognized as valid.

4.7.3 Subroutine UPDATE_PICTURE(oldpic,newpic, left,right,bottom,top,instructions,num_inst)

4.7.3.1 Purpose

This routine produces the graphic controller instructions necessary to convert one picture into another. It achieves this by comparing the new picture with the old picture, row by row, and converting the differences into a sequence of set and/or incremental move instructions. For example, if a user wishes to set up a sequence depicting a moving (singly-coloured) object, this routine would produce those instructions which would replace the trailing edge with the background, and replace the background with the leading edge. If it is known in advance that the differences in the two pictures are restricted to a particular (rectangular) region, the execution speed of this routine can be increased by restricting the operation of UPDATE_PICTURE to that region.

This routine can be used to generate a sequence of frames for display as a moving sequence with the MANPIX command MOVIE (see section 5.3.2).

4.7.3.2 Arguments

UPDATE_PICTURE has eight arguments:

(1) **oldpic**
Type: logical*1 array (0:511,0:511)
Use: This argument specifies the array containing the original picture which is to be updated, with each element specifying the colour of the corresponding pixel of the picture. Note that the dimensions of the array must be as specified, i.e., the whole screen must be included.

(2) **newpic**
Type: logical*1 array (0:511,0:511)
Use: This argument specifies the array containing the new picture which will result when the old picture is updated. This must also be dimensioned as specified.

(3) **left**
(4) **right**
(5) **bottom**
(6) **top**
Type: integer
Use: If it is known in advance that the differences in ol**dpic** and **newpic** are restricted to a particular rectangle of the screen, the speed of this routine can be increased by specifying the limits of this rectangle with these arguments. If such limits are unknown, or the differences are not restricted, specify **left**, **right**, **bottom** and top as 0, 511, 0 and 511 respectively.

(7) **instructions**
Type: integer*2 array or logical*2 array
Use: This argument specifies the array into which the sequence of instructions will be written. These instructions, when sent to the controller displaying **oldpic**, will transform the picture into **newpic**.

(8) **num_inst**
Type: integer
Use: This argument specifies the number of instructions written into the array **instructions**. It is therefore necessary that the **instructions** array be dimensioned sufficiently large in the calling program.

4.7.3.3 Errors

UPDATE_PICTURE detects no errors. However, no instructions will be generated if **left** is greater than **right**, or if **bottom** is greater than **top**.

4.7.4 Subroutine UPDATE_ROW (row,oldrow,newrow,instructions,num_inst)

4.7.4.1 Purpose

This routine produces the graphic controller instructions necessary to convert one row of pixels into another row of pixels. It works in exactly the same way as UPDATE_PICTURE, and is used when it is undesirable or impracticable to work on the whole picture at once.

4.7.4.2 Arguments

UPDATE_ROW has five arguments:

(1) **row**
Type: integer
Use: This argument specifies the row under consideration. It is required so that any set instructions produced will affect the correct row of the picture.

(2) **oldrow**
Type: logical*1 array (0:511)
Use: This argument specifies the array containing the original row to be updated, with each element specifying the colour of each pixel of the row. Note that the dimensions of the array must be as specified.

(3) **newrow**
Type: logical*1 array (0:511)
Use: This argument specifies the array containing the new row which will result when the old row is updated. This must also be dimensioned as specified.

(4) **instructions**
Type: integer*2 array or logical*2 array
Use: This argument specifies the array into which the sequence of instructions will be written. Note that this array must be dimensioned large enough in the calling program to hold the instructions generated.

(5) **num_inst**
Type: integer

Use: This argument specifies the number of instructions written into the array instructions.

4.7.5 Subroutine SEPARATE_UPDATE_ROW(row,oldrow,newrow, object_in_new, back_instr,back_ptr,object_instr,object_ptr)

4.7.5.1 Purpose

This routine produces the graphic controller instructions necessary to convert one row of pixels into another row of pixels. It works in a similar way to UPDATE_ROW except that the graphics instructions are placed into two output arrays - one for the background and one for an object. Thus, by sending the background sequence of instructions to the controller before the object instructions, the trailing edge of a "moving" object can be restored as background before the leading edge is drawn. This approach can, in some circumstances, overcome the problem of double images which may arise if background and object are drawn on a row by row basis (as would occur with the single instruction array generated by UPDATE_PICTURE or UPDATE_ROW). The separation of background and object is achieved with a user-supplied array which flags the object pixels in the new row.

4.7.5.2 Arguments

SEPARATE_UPDATE_ROW has eight arguments:

(1) **row**
Type: integer
Use: This argument specifies the row under consideration. It is required so that any set instructions produced will affect the correct row of the picture.

(2) **oldrow**
Type: logical*1 array (0:511)
Use: This argument specifies the array containing the original row to be updated, with each element specifying the colour of each pixel of the row. Note that the dimensions of the array must be as specified.

(3) **newrow**
Type: logical*1 array (0:511)
Use: This argument specifies the array containing the new row which will result when the old row is updated. This must also be dimensioned as specified.

(4) **object_in_new**
Type: logical*1 array (0:511)
Use: This argument is a flag array which specifies those pixels in the new row which contain the object of interest. These pixels are indicated by a value of 1, and instructions for these will go into the object output array. Background pixels are indicated by a value of 0, and their instructions will be placed into the background array. This must also be dimensioned as specified.

(5) **back_instr**
Type: integer*2 array or logical*2 array
Use: This argument specifies the array into which the sequence of instructions for updating the background part of the picture will be written. Note that this

array must be dimensioned large enough in the calling program to hold the instructions generated.

(6) **back_ptr**
Type: integer
Use: This argument specifies the number of instructions written into the array **back_instr**. It is a pointer to the last-used element of the array, and should be initialised by the user before this routine is called.

(7) **object_instr**
Type: integer*2 array or logical*2 array
Use: This argument specifies the array into which the sequence of instructions for updating the object part of the picture will be written. Note that this array must be dimensioned large enough in the calling program to hold the instructions generated.

(8) **object_ptr**
Type: integer
Use: This argument specifies the number of instructions written into the array **object_instr**. It is a pointer to the last-used element of the array, and should be initialised by the user before this routine is called.

4.7.6 Subroutine PIC_TO_INSTR(pic, left,right,bottom,top,instructions,num_inst)

4.7.6.1 Purpose

This routine produces the graphic controller instructions necessary to display a picture. The picture is processed on a row by row basis, and each row is stored as either a dump-mode restore instruction or as a group of set and incremental move instructions, according to whichever occupies the lesser amount of space. This routine thus produces the most efficient means of storing a picture.

4.7.6.2 Arguments

PIC_TO_INSTR has seven arguments:

(1) **pic**
Type: logical*1 array (left:right,bottom:top)
Use: This argument specifies the array containing the original picture which is to be converted to graphic instructions, with each element specifying the colour of the corresponding pixel of the picture. Note that the dimensions of the array must be as specified by the next four arguments of this routine.

(2) **left**
(3) **right**
(4) **bottom**
(5) **top**
Type: integer
Use: These arguments specify the dimensions of the picture to be converted to graphics instructions. Note that these are also the dimensions of the pic array.

(6) **instructions**
Type: integer*2 array or logical*2 array
Use: This argument specifies the array into which the sequence of instructions will be written. These instructions, when sent to a controller, will result in the original picture being displayed.

(7) **num_inst**
Type: integer
Use: This argument specifies the number of instructions written into the array **instructions**. It is therefore necessary that the **instructions** array be dimensioned sufficiently large in the calling program.

4.7.6.3 Errors

PIC_TO_INSTR detects no errors. However, no instructions will be generated if **left** is greater than **right**, or if **bottom** is greater than **top**.

4.7.7 Subroutine INSTR_TO_PIC(instructions,num_inst, picture[,clear[,error]])

4.7.7.1 Purpose

This routine "colours" the elements of a picture array according to a given sequence of graphic instructions, i.e., it operates on the picture array in the same way that the graphic controller operates on its memory, with the exception that instructions involving colour map quadrants are ignored, as are mask, synchronisation, scrolling and dump-mode saving instructions. INSTR_TO_PIC is the reverse of PIC_TO_INSTR. Note that this routine initialises colour and x and y coordinates to zero.

4.7.7.2 Arguments

INSTR_TO_PIC has five arguments, the last two being optional:

(1) **instructions**
Type: integer*2 array or logical*2 array
Use: This argument specifies the array containing the sequence of instructions that are to be written into the **picture** array (described below).

(2) **num_inst**
Type: integer
Use: This argument specifies the length of the **instructions** array.

(3) **picture**
Type: logical*1 array (0:511,0:511)
Use: This argument specifies the array which contains one element for each pixel on the screen. This array will be altered according to the instructions in the **instructions** array. Note that this array must be dimensioned as specified.

(4) **clear**
Type: logical
Use: This argument specifies whether or not the picture array is to be zeroed before applying the **instructions** array. If **clear** is omitted, it is assumed to be false, i.e., the picture array is not cleared first.

(5) **error**
Type: logical
Use: This argument is written by INSTR_TO_PIC, and is set to true if an instruction to write outside the screen area was detected. If this argument is omitted, no such indication of error is given.

4.7.7.3 Errors

INSTR_TO_PIC will return immediately to the calling program if an instruction to write outside the screen area was interpreted. If the error argument was specified, it will be set to true before INSTR_TO_PIC returns.

4.7.8 Subroutine FILL_ARRAY(array,num_elements,value)

4.7.8.1 Purpose

This routine employs the LIB$MOVC5 routine in the VAX run-time library to fill a byte array with a particular value. One use of this routine would be to clear out picture arrays, since it is far more efficient than a pair of "do" loops. FILL_ARRAY partitions the array into blocks of size 65535 or less, (the maximum size for LIB$MOVC5) and then calls LIB$MOVC5 to fill the array.

4.7.8.2 Arguments

FILL_ARRAY has three arguments:

(1) **array**
Type: logical*1 array
Use: This argument specifies the array whose elements are to be filled. Note that it is a byte array.

(2) **num_elements**
Type: integer
Use: This argument specifies the number of elements in array that are to be filled. **num_elements** should be not greater than the dimension of **array** in the calling program.

(3) **value**
Type: integer
Use: This argument specifies the value to be placed into each element of **array**. Note that only the least significant eight bits of **value** are used.

5. MANPIX USER'S GUIDE

MANPIX is a program which may be used for displaying and manipulating pictures on the graphic controller screens. It also allows a user to alter colour maps, read back parts of the picture, and to send individual instructions to the controllers.

MANPIX maintains the concept of "active screens" which the user defines with a UNIT command (section 5.1.2) and subsequent commands generally affect those screens which are currently active. MANPIX commands fall into three categories - those which are independent of the active screen environment, those which operate on all active screens with common parameters (or no parameters), and those where a separate parameter must be provided for each active screen. The common parameter commands result in repeated transmissions of the same group of graphic controller instructions - one for each screen, whereas the separate parameter commands generally transmit different groups of instructions.

To run MANPIX type:

MANPIX

to the "$". The program will respond with the prompt "Command:", to which a command string must be typed. Alternatively, the first command to MANPIX can be entered with the program name by typing

MANPIX <command-line>

to the "$". In that case, MANPIX will execute the requested command before issuing the prompt. Note that MANPIX converts all letters on input to upper case, so that commands can be entered in either case. For clarity, commands discussed below will be expressed in upper case.

5.1 Commands Independent of the Active Screen Environment

5.1.1 HELP Command

A listing of the available commands can be obtained at the user's terminal by entering the command HELP (either by typing MANPIX HELP to the "$" or by typing HELP to the "Command:" prompt). MANPIX will respond as follows:

```
     The following commands are available:

AFRAME      - show picture frame with "adaptive" map in file
BOX         - draw a box in a nominated colour
CLEAR       - clear the screen
COLUMN      - draw one or more columns in a nominated colour
DISPLAY     - display a number in decimal, octal and hexadecimal
DOTMOVE     - show a dot moving in a rectangular path
ECOLUMN     - read back part or all of a column from controller memory
EROW        - read back part or all of a row from controller memory
EXCHANGE    - exchange the pictures on two screens
FILLMAP     - fill up the colour map with a single value
FRAME       - show a picture frame stored as graphic instructions
HELP        - this message or more details on each command
INSTRUCTION - execute single graphic instructions
MAP         - load a colour map
MFRAME      - show multiple picture frame graphic instruction files
MOVIE       - show a sequence of frames as a "movie"
MSHOW       - show multiple pictures stored in files as pixel data
ROW         - draw one or more rows in a nominated colour
SCROLL      - scroll the picture
SET         - set all pixels to a nominated colour
SHOW        - show a picture stored in a file as pixel information
TRANSFER    - transfer a picture from one screen to another
UNIT        - select which screen units are to be active
VIEWMAP     - view the colours in the colour map
ZOOM        - zoom in on the picture currently being displayed
```

These commands can be abbreviated provided that enough is specified to distinguish between them. However, since more commands may be added at a future stage, it is recommended that a minumum of three characters is used.

Groups of commands can be assembled into a file (one command per line) and executed together by issuing the command "@filename", where filename is the name of the file containing the commands. Note that such a command file cannot contain another "@" command. It is recommended that the file type ".MPX" be used for MANPIX command files.

Parameters to commands must be separated by one or more spaces. Note that some commands have optional parameters - these will be shown in brackets ([]), with nesting of the brackets where appropriate.

Where integers are to be specified, octal constants can be entered by prefacing them with a letter "o", hex constants by prefacing with a letter "x".

Command format: HELP [command]
Parameters: command - the command about which information is sought

If command is omitted, the above description of all commands is given.

5.1.2 UNIT Command

This command is used to select which of the three controller screens are to be active. Note that when MANPIX is first entered, screen 1 (the centre screen) is the only active screen.

Command format: UNIT screen-1 [screen-2 [screen-3]]
Parameters: screen-n - one of 0, 1 or 2 to indicate a screen which is to become active.

One, two or all three screens can be currently active. The order in which they are specified in the UNIT command dictates the order in which changes will be made on each active screen. Note that the screens selected by this command remain active until another UNIT command is entered.

5.1.3 TRANSFER Command

This command transfers the picture from one screen to another, without affecting the picture on the first screen, resulting in two screens having the same picture in controller memory. It functions by reading the picture back from the first controller's memory, then transmitting the data to the second controller.

Command format: TRANSFER screen-1 screen-2
Parameters: screen-1 - the screen containing the picture to be transferred

screen-2 - the destination screen

Note that this command does not alter the current list of active screens.

5.1.4 EXCHANGE Command

This command exchanges the pictures on two of the screens. It functions by reading the pictures back from each of the selected controllers, and then transmitting the data to the opposite screens.

Command format: EXCHANGE screen-1 screen-2

Parameters: screen-n - the screens involved in the exchange

Note that this command does not alter the current list of active screens.

5.1.5 DISPLAY Command

This is a utility command to display one or more integers in decimal, octal and hexadecimal representation. The values are displaved on the user's terminal, not on the graphic screen.

Command format: DISPLAY n1 [n2 [n3 [n4 [n5 [n6 [n7 [n8 [n9 [n10]]]]]]]]]
Parameters: the numbers to be displayed (up to 10 can be entered at a time)

To enter a number in octal, prefix it with a letter "o".
To enter a number in hexadecimal, prefix it with a letter "x".

5.1.6 QUIT Command

This command causes MANPIX to exit to the VMS operating system.

Command format: QUIT
No parameters.

5.2 Common Parameter Commands

5.2.1 General

The commands in this category act sequentially upon the active controllers, in turn, and in the order that they were selected by the most recent UNIT command (section 5.1.2). Three of these commands, namely ECOLUMN (section 5.2.7), EROW (section 5.2.8) and ZOOM (section 5.2.18) require that only one screen be currently active; the others are valid for one, two or three active screen cases.

5.2.2 AFRAME Command

This command operates on a file which contains both graphic controller colour map information and picture information. It sets up the colour map of each of the active controllers, then displays the stored picture on each. The file must contain the picture information as a series of graphic instructions (rather than containing the colour of each pixel explicitly). This is generally the most efficient method of storing a picture, and results in the fastest subsequent display. The file must be set up as a FORTRAN binary file, with each frame being stored as one record in the following format (note the different length integers):

red(0:255)	logical*1 array
green(0:255)	logical*1 array
blue(0:255)	logical*1 array
n = number of instructions stored	integer*4
instructions(1:n)	integer*2 or logical*2 array

The three logical*1 arrays contain the red, green and blue components of the colour map, and should be written as a separate FORTRAN record from that containing the picture instructions. A typical FORTRAN sequence to produce the correct format would be as follows:

```
logical*1 red(0:255),green(0:255),blue(0:255)
integer*4 n
logical*2 instr(10000)

   (code to generate map and instructions)

write(1) red,green,blue
write(1) n,(instr(i),i=1,n)
```

Command format: AFRAME filename [frame-number]

Parameters: filename - the name of the binary file containing the frame(s) of colour maps and graphic instructions

frame-number - the frame number in the file to be shown (first frame is number 1)

If frame-number is omitted, it is assumed to be 1.

5.2.3 BOX Command

This command will draw a rectangle on each of the active screens in a nominated colour.

Command format: BOX colour left right bottom top

Parameters:
- colour - number in colour map for the box's colour
- left - pixel number of left-hand edge
- right - pixel number of right-hand edge
- bottom - row number of bottom edge
- top - row number of top edge

5.2.4 CLEAR command

This command clears each of the active screens by setting all pixels of each controller to colour zero. It is assumed that colour zero is set up as black in the colour map. Note that CLEAR is equivalent to SET 0 (the SET command is described below in section 5.2.15).

Command format: CLEAR
No parameters.

5.2.5 COLUMN command

This command fills one or more columns of each of the active screens in a nominated colour. All 512 pixels of each column are coloured.

Command format: COLUMN colour start-pixel [finish-pixel]

Parameters:
- colour - number in colour map for column colour
- start-pixel - pixel number of the left-most column
- finish-pixel - pixel number of the right-most column

If finish-pixel is omitted, one column is coloured at the start-pixel location.

5.2.6 DOTMOVE Command

This command causes a dot to move in a rectangular path on each of the active screens.

Command format: DOTMOVE colour left right bottom top

Parameters:
- colour - number in colour map for the dot's colour
- left - pixel number of left-hand edge of path
- right - pixel number of right-hand edge of path
- bottom - pixel number of bottom edge of path
- top - pixel number of top edge of path

Note: the dot keeps moving until a carriage return is typed.

5.2.7 ECOLUMN Command

This command enables a user to examine part or all of a column of pixels by reading back from the graphic controller's memory. Note that only one screen can be active for this command to execute successfully - if there is

more than one screen currently active an error message will be given and MANPIX will request another command.

```
Command format: ECOLUMN column-number start-row [finish-row]
Parameters:     column-number - pixel number of column to be examined
                start-row     - first row of the column to be read back
                finish-row    - last row of the column to be read back
```

All pixels in the column between start-row and finish-row (inclusive) are read back from the controller's memory and displayed at the user's terminal. If the user requires the data on a file rather than at the terminal, then the logical name FOR$TYPE should be directed to a file before running MANPIX (e.g. by issuing the command DEFINE/USER FOR$TYPE filename.typ). Note that colour values are given in hexadecimal to reduce space requirements. However, conversion of values to octal or decimal can be achieved with the DISPLAY command (see section 5.1.5).

If finish-row is omitted, one pixel only is read back - that pixel whose screen coordinates are (column-number,start-row).

5.2.8 EROW Command

This command enables a user to examine part or all of a row of pixels by reading back from the graphic controller's memory. It is the same as the ECOLUMN command (section 5.2.7) except that it reads horizontally rather than vertically. Note that only one screen can be active for this command to execute successfully - if there is more than one screen currently active an error message will be given and MANPIX will request another command.

```
Command format: EROW row-number start-pixel [finish-pixel]
Parameters:     row-number   - row (y-coordinate) to be examined
                start-pixel  - first pixel of the row to be read back
                finish-pixel - last pixel of the row to be read back
```

All pixels in the row between start-pixel and finish-pixel (inclusive) are read back from the controller's memory and displayed at the user's terminal. If the user requires the data on a file rather than at the terminal, then the logical name FOR$TYPE should be re-directed as mentioned above in the ECOLUMN command (section 5.2.7). As in the case of the ECOLUMN command, values are given in hexadecimal.

If finish-pixel is omitted, one pixel only is read back - that pixel whose screen coordinates are (start-pixel,row-number).

5.2.9 FILLMAP Command

This command is used to fill all 256 colours in the map of each active controller with a single value. Note that the current colour map quadrant is used. If a different quadrant is required, it can be selected using the INSTRUCTION command (section 5.2.11) or the VIEWMAP command (section 5.2.17).

```
Command format: FILLMAP red green blue
Parameters:     red   - value of red component to go in the map
                green - value of green component to go in the map
                blue  - value of blue component to go in the map
```

5.2.10 FRAME Command

This command displays on each active screen a picture stored in a file as a series of graphic instructions (rather than storing explicitly the colour of each pixel). It is the same as the AFRAME command (section 5.2.2) except that the file contains no information about the colour map. The file must be set up as a FORTRAN binary file, with each frame being stored as one record in the following format:

n = number of instructions stored	integer*4
instructions(1:n)	integer*2 or logical*2 array

Command format: FRAME filename [frame-number]
Parameters: filename - the name of the binary file containing the frame(s) of graphic instructions

frame-number - the frame number in the file to be shown (first frame is number 1)

If frame-number is omitted, it is assumed to be 1.

5.2.11 INSTRUCTION Command

This command enables a user to send individual intstructions to all graphic controllers that are currently active. All instructions except the dump-mode save and restore instructions can be transmitted.

Command format: INSTRUCTION code parameter-list
Parameters: code - a 1 or 2 letter code representing the instruction to be sent (see below)

parameter-list - required parameters for the instruction

The following instructions can be sent:

code	parameter-list
i (incremental move)	x-move y-move repeat-count
l (load colour map word)	address red green blue
x (set x coordinate)	x-value
y (set y coordinate)	y-value
xw (set x & write pixel)	x-value
yw (set y & write pixel)	y-value
c (set colour)	colour
cw (set colour & write all pixels)	colour
m (set mask)	mask-value
qw (select write quadrant)	quadrant-number
qd (select display quadrant)	quadrant-number
s (scroll)	x-move y-move
sr (scroll reset)	<none>
d (disable picture display)	<none>

5.2.12 MAP Command

This command is used to load, into each controller that is currently active, a colour map which has been set up previously in a FORTRAN binary file. Note that the current colour map quadrant is used. If a different quadrant is required, it can be selected using the INSTRUCTION command (section 5.2.11) or the VIEWMAP command (section 5.2.17). The MAP command requires that the FORTRAN binary file be set up as one record in the format used by subroutine LOAD_MAP_FILE of the FLISGRAPH package (see section 4.3.8).

Command format: MAP filename
or MAP ?
Parameters: filename - the name of the colour map file

If filename is specified, the MAP: directory is searched for the file. If that search fails, the user's directory is searched, and if that fails an error results. If the file is found, the colour map is loaded.

If ? is specified, the user is given the names of the colour map files in the MAP: directory (one by one), and asked if that file's map is to be loaded. Once the user chooses a file, that map is loaded. If no choice is made, no action is taken.

5.2.13 ROW command

This command fills one or more rows of each active screen in a nominated colour. All 512 pixels of each row are coloured. This is similar to the COLUMN command (section 5.2.5), except that it works horizontally rather than vertically.

```
Command format: ROW colour start-row [finish-row]
Parameters:     colour     - number in colour map for the column colour
                start-row  - number of the lowest row to be coloured
                finish-row - number of the highest row to be coloured
```

If finish-row is omitted, one row is coloured at the start-row location.

5.2.14 SCROLL command

The SCROLL command causes the picture on each active screen to be shifted horizontally, vertically, or both. Wrap-around takes place, so, for example, if the picture is scrolled 2 pixels in the +X direction, all columns of the picture will move 2 columns to the right, with columns 0 and 1 being replaced by the former contents of columns 510 and 511.

```
Command format: SCROLL x-moves [y-moves]
            or  SCROLL reset
Parameters:     x-moves - the number of pixels of horizontal movement
                y-moves - the number of pixels of vertical movement

            or  reset   - instruction to reset the scroll offset
```

If y-moves is omitted, it is assumed to be zero.

If reset is specified, any previous scroll that was issued is cancelled, and the picture returns to its original position. In specifying reset, it can be abbreviated to 1 letter.

5.2.15 SET Command

This command alters all pixels on each active screen to a nominated colour. The CLEAR command (section 5.2.4) is a special case of this, in which the nominated colour is zero.

```
Command format: SET colour
Parameters:     colour - number in colour map for the screen colour
```

5.2.16 SHOW Command

This command is used to display, on each active screen, a picture stored in a file which contains the colour map number for each pixel of interest. The SHOW command allows the picture to be translated horizontally and/or vertically before it is displayed.

Note that for large pictures, this may not be the most efficent use of storage - a full picture would require over 500 disk blocks if stored in this way. A more compact form may be achieved by converting the pixel colour information into set and incremental move instructions. This can be done using subroutine PIC_TO_INSTR in the FLISGRAPH package (see section 4.7.6) or by using the ZOOM command (see section 5.2.18). The subsequent picture can be displayed from its new file with the FRAME command (see section 5.2.10).

Use of the SHOW command requires that the file be set up as a FORTRAN binary file. The picture to be displayed must be a rectangle within the screen (or the whole screen), and information for each pixel within the rectangle must be stored in the FORTRAN binary file as one record set out in the following format:

```
left   = pixel number of the rectangle's left edge    integer*4
right  = pixel number of the rectangle's right edge   integer*4
bottom = row number of the rectangle's bottom edge    integer*4
top    = row number of the rectangle's top edge       integer*4
picture(left:right,bottom:top)                        logical*1 array
```

Note that the dimensions of the two-dimensional picture array must correspond with the rectangle edges.

Command format: SHOW filename [x-offset [y-offset]]

Parameters:
- filename - the name of the binary file containing the picture
- x-offset - the number of pixels that the picture is to be displaced horizontally before showing
- y-offset - the number of pixels that the picture is to be displaced vertically before showing

If offset values are omitted, they are assumed to be zero. Note that an x-offset must be specified if a y-offset is to be specified.

5.2.17 VIEWMAP Command

This command allows a user to see the colours in the map of each active controller by displaying a 16 by 16 array of rectangles - one for each colour. Each rectangle is labelled in colour zero with the colour number of the rectangle. The VIEWMAP command allows the user to select a particular quadrant for the displaying of its map. Note that selecting a quadrant alters the current colour map quadrant to that value.

Command format: VIEWMAP [quadrant-number]

Parameter: quadrant-number - the number of the quadrant whose colour map is to be viewed. The current quadrant-number is left at this value.

If quadrant-number is omitted, the current quadrant is used.

5.2.18 ZOOM Command

The ZOOM command allows a user to "zoom" in on the picture which is currently on the screen. The user can select a point on the screen to be the new picture centre, and then specifiy a magnification, M. The picture is then displayed with one pixel of the original picture being represented by an M by M rectangular array of pixels. Note that only one screen can be active for this command to execute successfully - if there is more than one screen currently active an error message will be given and MANPIX will request another command.

Command format: ZOOM

This command has no parameters - a dialogue is conducted with the user to determine requirements. When MANPIX ZOOM is typed to the "$", or just ZOOM to the MANPIX prompt "Command:", the program types the following message:

Reading back the picture...

This operation takes 10 to 15 seconds. Once the picture has been read, MANPIX draws a pair of cross hairs in the centre of the screen. The user will be asked to enter commands which will move the cross to the desired centre of the enlarged picture. The cross is shown in colour zero. However, this may be unsuitable if the picture consists of a small coloured region with a large black background, because the cross will be indistinguishable. Hence, MANPIX asks the following question first:

Cross colour ok? (Type Y or new colour number):

At this point the user either enters Y or a new colour number. If a new number is entered, the question is asked again. Once the user has entered Y, the following is asked

Move cross to new centre (Up, Down, Left, Right, or 0)?

The user can now enter commands to re-locate the cross to the desired centre. Such commands consist of one of the letters U, D, L, or R optionally followed by a number. These will move the cross the stated number of pixels up, down, left or right. If the number of pixels is omitted, it is assumed to be 1. If a command moves the cross beyond a screen boundary, it is placed at that boundary. Each time the user enters a command, the cross is shifted and a shortened form of the question is asked:

Move cross?

The short form is used from then onwards unless an invalid command is entered when the long form is re-used. The user should continue this procedure until the cross is at its desired location, and then answer the question with 0 (zero). MANPIX will then respond:

Enter magnification (0 if done):

The user can then enter the desired magnification. MANPIX will display the enlarged picture, and then ask again for magnification. When the user has finished looking at the picture at various magnifications, a zero should be entered. MANPIX will then ask

Now what - S(ave on file, M(ore of this, E(xit?

The user should respond with S, M, or E. M will return to the cross hairs level, so that the same picture can be enlarged about a different centre. E will cause MANPIX to return to the command level. S will result in the following request:

Enter filename:

The user should enter a valid VAX Files-11 filename and MANPIX will then convert the current picture into a sequence of graphic instructions and store them in that file. This picture can subsequently be re-displayed with the FRAME command (see section 5.2.10). When the data has been stored, MANPIX will re-ask the question seeking S, M, or E as a response.

The ZOOM command can be used to convert a file stored as pixel information (to be displayed with the SHOW command - section 5.2.16) into a file stored as a sequence of graphic instructions (to be displayed with the FRAME command - section 5.2.10). This can be achieved by displaying the picture, and entering the ZOOM command. Enter Y to the "Cross colour ok?" question, and 0 to the "Move cross?" question. Then enter a magnification of 1, S to the "Now what" question and give a valid filename when MANPIX

asks for it. The resulting file will contain the graphic instructions necessary to generate that picture from a clear screen.

5.3 Separate Parameter Commands

5.3.1 MFRAME Command

This command displays on each active screen a picture stored in a file as a series of graphic instructions. A separate file must be specified for each screen that is currently active. It is similar to the FRAME command (section 5.2.10) except that no frame-number can be specified - this command only displays the first frame of each file. Each file must be set up as a FORTRAN binary file as described in section 5.2.10.

Command format:	MFRAME file-1	(1 active screen)
or	MFRAME file-1 file-2	(2 active screens)
or	MFRAME file-1 file-2 file-3	(3 active screens)
Parameters:	file-n - the name of the binary file containing the frame(s) of graphic instructions for the n-th screen in the list of those currently active	

Note that "MFRAME file file file" has the same effect as "FRAME file" when all three screens are active; similar command pairs are equivalent when there are one or two screens active.

5.3.2 MOVIE Command

This command is used to show a sequence of picture frames stored in a file as a moving sequence. A file must be supplied for each screen that is currently active, and each frame of each file must be set up as one record of a FORTRAN binary file in exactly the same format as required for the FRAME command (see section 5.2.10). The instructions necessary for converting one picture frame into another can be generated using subroutine UPDATE_PICTURE of the FLISGRAPH package (see section 4.7.3).

The rate at which the frame sequence is updated can be specified by the user as a multiple of 10 milliseconds. Alternatively, the user can request that the display of the graphic instructions be delayed by the controller so that the picture is synchronized with either the frame (25 Hz) or field (50 Hz) update. Synchronization with the frame update is the recommended way of showing a moving sequence at 25 frames per second.

Command format:	MOVIE file-1 [sync-or-interval]
or	MOVIE file-1 file-2 [sync-or-interval]
or	MOVIE file-1 file-2 file-3 [sync-or-interval]

Parameters: file-n - the name of the binary file containing the frames of graphic instructions for the n-th screen in the list of those currently active

sync-or-interval - time interval or synchronization specification

(i) interval - the time interval between updates from one frame to the next. Note that this, if specified, must be expressed in millisec, and must be a multiple of 10 millisec.

(ii) sync - must be either "field" or "frame" and if specified, transfer of the graphic instructions to the controller will be delayed accordingly.

If sync-or-interval is omitted or specified as 0, the sequence will be shown as fast as possible. Note that the sequence is shown over and over again until the user enters a carriage return.

If a requested update rate is too high (because other users are making heavy demands on the system, or because there is a very large number of graphic instructions per frame to transmit) an error condition will result. MANPIX will display an error message and return to command level. The error message will be of the form:

```
*** interval too short - overrun in pass no. ppppp frame no. fffff
```

where ppppp is the number of the current iteration through the moving sequence and fffff is the number of the frame whose transmission caused the overrun.

5.3.3 MSHOW Command

This command displays on each active screen a picture stored in a file as pixel information. A separate file must be specified for each screen that is currently active. This command is similar to the SHOW command (section 5.2.16) except that offsets cannot be supplied to translate the picture. The file format is the same as that described in section 5.2.16.

Command format: MSHOW file-1
<u>or</u> MSHOW file-1 file-2
<u>or</u> MSHOW file-1 file-2 file-3
Parameters: file-n - the name of the binary file containing the pixel information for the n-th screen in the list of those currently active

Note that "MSHOW file file file" has the same effect as "SHOW file" when all three screens are active; similar command pairs are equivalent when there are one or two screens active.

REFERENCE

[1] Sandor, J. and Lester, L.N.: "Computer Graphics for Air Traffic Control Tower Simulation", Proc. Conference on Computers and Engineering, 1983, Sydney, pp.24-29.

APPENDIX A

INSTRUCTION SET FOR RASTER GRAPHIC CONTROLLERS

A.1 General

This appendix details the instruction set of the raster graphics controllers. Instructions are sixteen bits, except for the "load colour map word" instruction which is 32 bits (expressed as two 16 bit words).

The following sections give a brief description and the bit pattern for each instruction. Note that in some cases, not all of the 16 bits are used - in these cases, the unused bits are shown as asterisks ("*") in the bit pattern. All bit patterns show the most significant bit at the left.

A.2 Set Colour Code

This instruction changes the value in the controller's colour code register to the value specified in its least significant 8 bits. This colour is used in all future instructions which write into the raster memory (until changed by another "set colour code" instruction). A bit is provided in this instruction as a flag to indicate whether or not all pixels are to be set to the new colour. Thus, the screen can be cleared with this instruction by setting all pixels to colour 0 (provided that colour 0 has been set up as black in the colour map).

Bit pattern: 0 0 1 0 S * * * A A A A A A A A

where

AAAAAAAA (bits 0-7) = colour (address in the colour map (0-255))
S (bit 11) = flag to set all pixels to the new colour

A.3 Set X Coordinate

This instruction modifies the controller's x coordinate holding register to the value specified in its least significant 9 bits, and together with the "set y coordinate" instruction, allows individual pixels to be referenced. A bit is provided as a flag to indicate whether or not the pixel being referenced (by the updated x and y coordinate holding registers) is to be set to the prevailing colour (as set up with the most recent "set colour code" instruction).

Bit pattern: 0 0 0 0 W * * X X X X X X X X X

where

XXXXXXXXX (bits 0-8) = new value for the x coordinate
W (bit 11) = flag to set referenced pixel to the prevailing colour

A.4 Set Y Coordinate

This instruction modifies the controller's y coordinate holding register in the same manner that the "set x coordinate" instruction modifies the x coordinate holding register. As in the case of that instruction, a bit is provided to indicate whether or not the referenced pixel is to be set to the prevailing colour.

Bit pattern: 0 0 0 1 W * * Y Y Y Y Y Y Y Y Y

where

YYYYYYYYY (bits 0-8) = new value for the y coordinate
W (bit 11) = flag to set referenced pixel to the prevailing colour

A.5 Set Mask

This instruction sets the controller's mask register. This register is used to determine which of the controller's eight memory planes can be written. Setting one of the mask bits to zero prevents modification of the corresponding bit plane, with the result that only a subset of the full 256 colours may be written.

Bit pattern: 0 0 1 1 0 * * * M M M M M M M M

where

MMMMMMMM (bits 0-7) = new value for the mask

A.6 Select Write Colour Map Quadrant

As discussed in section 2, the graphic controller has four colour map quadrants, with each storing the red, green and blue components of a set of 256 colours. This instruction is used to select a particular quadrant so that one or more of its 256 colours can be altered (with a "load colour map word" instruction).

Bit pattern: 0 1 0 0 * * * * * * * * * * Q Q

where

QQ (bits 0-1) = quadrant number to be selected (0-3)

A.7 Select Display Colour Map Quadrant

This instruction selects the colour map quadrant to be used in translating the 256 colours of the map into red, green and blue values for display. A bit is provided in this instruction for enabling and disabling the display. Thus, if this instruction is sent with this bit clear, the screen remains blank until another "select display colour map quadrant" instruction is sent with this bit set.

Bit pattern: 0 1 0 1 E * * * * * * * * * Q Q

where

QQ (bits 0-1) = quadrant number to be selected (0-3)
E (bit 11) = flag to enable display

A.8 Load Colour Map Word

This instruction requires 32 bits, and is used to define the red, green and blue values for an entry in the colour map. There are 7 bits for red, 8 bits for green, and 6 bits for blue. There is also one bit provided to

indicate whether or not this colour will be detectable by a light pen. The quadrant used is that selected by the most recent "set write colour map quadrant" instruction.

```
Bit pattern:     1 1 B B B B B B A A A A A A A A
                 L R R R R R R R G G G G G G G G
```

where

```
     AAAAAAAA   (bits 0-7) = address in the colour map (0-255)
       BBBBBB  (bits 8-13) = blue level for this colour (0-63)
     GGGGGGGG (bits 16-23) = green level for this colour (0-255)
      RRRRRRR (bits 24-30) = red level for this colour (0-127)
            L     (bit 31) = flag to make this colour detectable by a
                             light pen
```

A.9 Incremental Move

This instruction alters the controller's x and y coordinate registers by a requested number of steps in any of the eight cardinal directions. A bit is provided to indicate whether or not pixels traversed in the "move" are to be set to the prevailing colour (as set up with the most recent "set colour code" instruction). Thus, this instruction is most useful for filling in all or part of a row, column or diagonal line of pixels.

```
Bit pattern:     1 0 W X L Y D C C C C C C C C C
```

where

```
    CCCCCCCCC   (bits 0-8) = repeat count - number of steps to move
                             (0-511)
            D      (bit 9) = flag to indicate y move is negative
                             (downwards)
            Y     (bit 10) = flag to request a y-directional move
            L     (bit 11) = flag to indicate x move is negative
                             (leftwards)
            X     (bit 12) = flag to request an x-directional move
            W     (bit 13) = flag to indicate that all pixels traversed
                             are to be set to the prevailing colour
```

A.10 Scroll

This instruction causes the picture on the screen to be scrolled by one pixel in any of the eight cardinal directions. The picture wraps around so that, for example, column 511 becomes column 0 if a scroll in the positive x direction is performed.

```
Bit pattern:     0 1 1 0 * * * * * * * 0 X L Y D
```

where

```
            D      (bit 0) = flag to indicate y scroll is negative
                             (downwards)
            Y      (bit 1) = flag to request a one pixel scroll in the
                             y direction
            L      (bit 2) = flag to indicate x scroll is negative
                             (leftwards)
```

X (bit 3) = flag to request a one pixel scroll in the x direction

A.11 Scroll Reset

This instruction resets the x and y scroll offsets which have been set up with any previous scroll instructions. Thus, a "scroll reset" instruction will cause a previously scrolled picture to return to its original position on the screen.

Bit pattern: 0 1 1 0 * * * * * * * * 1 * * * *

A.12 Synchronise Controller

This instruction causes the transmission of all following instructions to be delayed, so that a picture update can be synchronised with the field or frame blanking interval of the display monitor. A bit is provided to indicate whether transmission is to be delayed until the next field blank (start of scan line 295 or 607) or until the next frame blank (start of scan line 607).

Bit pattern: 0 0 1 1 1 * * * * * * * * * * * F

where

F (bit 0) = flag to indicate whether transmission is to be delayed until field blank (clear) or frame blank (set)

A.13 Save

This instruction allows part or all of a row or column of pixels to be copied from the controller's memory planes into the host LSI's memory. The copying starts at the pixel represented by the current controller x and y coordinates, with N pixels being copied, where N is one more than the count in the least significant 9 bits of the instruction. Each pixel is stored as a byte in the LSI memory starting at the next location following this "save" instruction. A bit is provided to indicate whether the copy is to be performed across a row (x direction) or up a column (y direction). The copying always takes place in the positive direction.

Bit pattern: 0 1 1 1 0 P * N N N N N N N N N

where

NNNNNNNNN (bits 0-8) = ONE LESS THAN the number of pixels to save (0-511 representing 1-512 pixels)

P (bit 11) = flag to indicate path of save - clear for x direction, set for y direction

A.14 Restore

This instruction is the same as the "save" instruction except that the transfer of pixel data is from the host LSI's memory to the graphic controller's memory. Thus, this instruction can be used to restore part or all of

a row or column of pixels previously stored with the "save" instruction. Each pixel is retrieved from a byte in the LSI memory starting at the next location following this "restore" instruction.

Bit pattern: 0 1 1 1 1 P * N N N N N N N N N

where

NNNNNNNNN	(bits 0-8) =	ONE LESS THAN the number of pixels to be restored (0-511 representing 1-512 pixels)
P	(bit 11) =	flag to indicate path of restore - clear for x direction, set for y direction

APPENDIX B

PRIMARY BOOTSTRAP FOR USE WITH PROGRAM LSILOAD

The primary bootstrap is as follows, and should be loaded into the host LSI starting at address 157720 (octal). Note that all numbers in this appendix are octal integers.

```
157720:  12701        MOV  #157534,R1
157722: 157534
157724:  12702        MOV  #2,R2
157726:      2
157730: 113700    1$: MOVB @#DRBCSR+1,R0
157732: 172425
157734: 113703        MOVB @#DRBCSR,R3
157736: 172424
157740:  74300        XOR  R3,R0
157742:  30200        BIT  R2,R0
157744:   1771        BEQ  1$
157746:  13721        MOV  @#DRBINP,(R1)+
157750: 172426
157752:  74237        XOR  R2,@#DRBCSR
157754: 172424
157756:    764        BR   1$
```

The interface between the VAX-11/780 and the LSI-11/23 comprises a DR11-W (on the VAX) coupled to a DRV11-B (on the LSI). These form the DMA link between the VAX and LSI. For program loading, however, the DR11-W and DRV11-B are used solely as a register interface, so that the primary bootstrap can be as simple as possible. This is because the primary bootstrap has to be keyed in by hand on some LSI microcomputers.

In the code above, DRBCSR is the address of the DRV11-B control/status register (172424 octal) and DRBINP is the DRV11-B input buffer register (172426 octal). The DR11-W and DRV11-B are coupled such that the input buffer register of each is the output buffer register of the other, and vice versa. Communication is achieved via the two bits in the control/status register (CSR) termed the "FUNCT1" and "STATC" bits. When the FUNCT1 bit is altered in one CSR, the STATC bit changes accordingly in the other CSR.

The VAX sends data to the LSI using the following algorithm:

Step 1. Wait for the FUNCT1 and STATC bits in the CSR of the DR11-W to be the same.

Step 2. Write data into the output buffer register of the DR11-W.

Step 3. Alter the FUNCT1 bit in the CSR of the DR11-W.

The LSI receives data from the VAX using the following algorithm:

Step 1. Wait for the FUNCT1 and STATC bits in the CSR of the DRV11-B to be different.

Step 2. Copy data from the input buffer register of the DRV11-B into memory.

Step 3. Alter the FUNCT1 bit in the CSR of the DRV11-B.

DISTRIBUTION

AUSTRALIA

Department of Defence

Central Office

Chief Defence Scientist)
Deputy Chief Defence Scientist) (1 copy)
Superintendent, Science and Program Administration)
Controller, External Relations, Projects and
Analytical Studies)
Defence Science Adviser (U.K.) (Doc Data sheet only)
Counsellor, Defence Science (U.S.A) (Doc Data sheet only)
Defence Science Representative (Bangkok)
Defence Central Library
Document Exchange Centre, D.I.S.B. (18 copies)
Joint Intelligence Organisation
Librarian H Block, Victoria Barracks, Melbourne
Director General - Army Development (NSO) (4 copies)

Aeronautical Research Laboratories

Director
Library
Superintendent Systems
Divisional File - Systems
Author: L. N. Lester (5 copies)
Group Leader Flight Simulation Group

Materials Research Laboratories

Director/Library

Defence Research Centre

Library

Navy Office

Navy Scientific Adviser

Army Office

Army Scientific Adviser
Royal Military College Library

Air Force Office

Air Force Scientific Adviser
Aircraft Research and Development Unit
Scientific Flight Group
Library
Technical Division Library
RAAF Academy, Point Cook

DISTRIBUTION (CONT'D)

Central Studies Establishment

Information Centre

Department of Defence Support

Government Aircraft Factories

Manager
Library

UNITED KINGDOM

Royal Aircraft Establishment
Bedford, Library
British Library, Lending Division

UNITED STATES OF AMERICA

NASA Scientific and Technical Information Facility

SPARES (10 copies)

TOTAL (63 copies)

Department of Defence

DOCUMENT CONTROL DATA

1. a. AR No	1 b. Establishment No	2. Document Date	3. Task No
AR-003-939	ARL-SYS-TM-72	JUNE, 1984	DST 82/069

4. Title	5. Security a. document	6. No Pages
FLIGHT SYSTEMS RASTER GRAPHICS SOFTWARE REFERENCE MANUAL	UNCLASSIFIED	53
	b. title U c. abstract U	7. No Refs 1

8. Author(s)	9. Downgrading Instructions
LEWIS NEALE LESTER	

10. Corporate Author and Address	11. Authority (as appropriate)
Aeronautical Research Laboratories, Box 4331, P.O. Melbourne Vic 3001	a.Sponsor b.Security c.Downgrading d.Approval

12. Secondary Distribution (of this document)

Approved for Public Release

Overseas enquirers outside stated limitations should be referred through ASDIS, Defence Information Services Branch, Department of Defence, Campbell Park, CANBERRA ACT 2601

13. a. This document may be ANNOUNCED in catalogues and awareness services available to ...

No limitations

13. b. Citation for other purposes (ie casual announcement) may be (select) unrestricted (or) as for 13 a.

14. Descriptors	15. COSATI Group
Computer graphics Raster graphic devices Software Computer program documentation	09020

16. Abstract

Flight simulation work at Aeronautical Research Laboratories employs raster graphics for both the presentation of visual scenes with limited dynamic content and for the replication of both flight instruments and sensor displays. The raster graphics system consists of an LSI-11/23 microprocessor (functioning as host) and three A.R.L. designed controller units coupled to RGBS monitors, and is connected through the LSI-11/23 to the Flight Simulation Group VAX-11/780 computer. This Memorandum describes the raster graphic system, and four software packages which are available to enable a VAX user to create a manipulate pictures. These packages are: RJRLSI which runs in the LSI-11/23 for communication; LSILOAD, for loading

This page is to be used to record information which is required by the Establishment for its own use but which will not be added to the DISTIS data base unless specifically requested.

16. Abstract (Contd)

and executing LSI programs; FLISGRAPH, a set of subroutines for creating and manipulating pictures; and MANPIX, a utility program which provides a versatile range of picture manipulation capabilities through a simple set of commands.

17. Imprint

Aeronautical Research Laboratories

18. Document Series and Number	19. Cost Code	20. Type of Report and Period Covered
Systems Technical Memorandum 72	716150	

21. Computer Programs Used

RJRLSI
LSILOAD
FLISGRAPH
MANPIX
FORTRAN
MACRO-11
VAX-11 MACRO

22. Establishment File Ref(s)